ARI Research Note 87-26

Knowledge Representation in the PUPS Theory

John R. Anderson
Ross Thompson
Carnegie Mellon University

for

Contracting Officer's Representative
Judith Orasanu

BASIC RESEARCH LABORATORY
Milton R. Katz, Director

U. S. Army
Research Institute for the Behavioral and Social Sciences

May 1987

U. S. ARMY RESEARCH INSTITUTE

FOR THE BEHAVIORAL AND SOCIAL SCIENCES

A Field Operating Agency under the Jurisdiction of the

Deputy Chief of Staff for Personnel

EDGAR M. JOHNSON
Technical Director

WM. DARRYL HENDERSON
COL, IN
Commanding

Research accomplished under contract
for the Department of the Army

Carnegie-Mellon University

Technical review by

Steve Kronheim

Accession For		
NTIS GRA&I	☒	
DTIC TAB	☐	
Unannounced	☐	
Justification		
By		
Distribution/		
Availability Codes		
Dist	Avail and/or Special	
A-1		

QUALITY INSPECTED 4

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		**READ INSTRUCTIONS BEFORE COMPLETING FORM**
1. REPORT NUMBER ARI Research Note 87-26	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Knowledge Representation in the PUPS Theory		5. TYPE OF REPORT & PERIOD COVERED Interim Report June - December 1986
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) John R. Anderson and Ross Thompson		8. CONTRACT OR GRANT NUMBER(s) MDA 903-85-K-0343
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Psychology Carnegie-Mellon University Pittsburgh, PA 15213-3890		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 2Q161102B74F
11. CONTROLLING OFFICE NAME AND ADDRESS U.S. Army Research Institute for the Behavioral and Social Sciences, 5001 Eisenhower Avenue, Alexandria, VA 22333-5600		12. REPORT DATE May 1987
		13. NUMBER OF PAGES 51
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) - -		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE - -

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release, distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

- -

18. SUPPLEMENTARY NOTES

Judith Orasanu, contracting officer's representative

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Knowledge Representation,	Problem-Solving,	Linguistics,
Procedural Knowledge,	Communication,	~~Cognition~~
Declarative Knowledge,	PUPS Analogy	Syntax,
Cognitive Skills,	ACT* Theory,	LISP.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This research note is concerned with the PUPS theory of cognition. PUPS was developed from the ACT* theory in response to evidence about the importance of analogy to cognition. This report follows a general discussion of representation with a discussion of the analogy mechanism which drives the PUPS representation. A formal specification of the PUPS representation is next presented, and finally the connection between this representation and earlier propositional representations of the ACT* theory is discussed. [illegible]

This is a paper with two functions. Its primary function is to present a new theory of knowledge representation which is the foundation for the PUPS (Anderson & Thompson, in press) theory of cognition. The second is to use the development of this theory as a case for discussing the role of knowledge representation in a cognitive psychology theory. The organization of this paper will reflect this dual function. It will begin with a discussion of issues of knowledge representation. Then we will turn to knowledge representation in the PUPS theory.

PUPS evolved from ACT* (Anderson, 1983) in response to evidence about the importance of analogy to cognition. The knowledge representation is designed to permit development of the analogy process and to permit analysis of it. Therefore, the paper will follow the general discussion of representation with a discussion of the analogy mechanism that has driven the PUPS representation. Then we will present a formal specification of the PUPS representation. Finally, we will close with a discussion of the connection between this representation and the earlier propositional representations of the ACT theory (Anderson, 1976; 1983).

The Issue of Knowledge Representation

One of the major developments in cognitive psychology over its behaviorist predecessors is the emphasis it has given to representation of knowledge. The basic idea was that not only is it important that we postulate internal states but it is important how we represent these internal states. Thus, for instance, the propositional-imagery debate (Anderson, 1978; Kosslyn & Pomerantz, 1977; Pylyshyn, 1973) was not a debate over the issue of internal states. It was a debate over the issue of how to represent these internal states.

Researchers' experience in developing cognitive models had convinced them that internal representation was absolutely critical. One example concerns the representation of linear arrays. Anderson & Bower (1973) had proposed that we represent a list like "monkey tiger giraffe cow donkey" as a series of propositions of the form

1. (monkey left-of tiger)
2. (tiger left-of giraffe)
3. (giraffe left-of cow)
4. (cow left-of donkey)

This led to the prediction that it would take subjects longer to make an inference like "the tiger is left of the donkey" where they had to combine a number of these pairwise orderings than to verify a sentence like "the tiger is left of the giraffe" where they did not. No sooner had we put this prediction to ink than Potts (1972) published the first experiments showing that just the opposite was true -- subjects judgements were faster the further the terms were away. We were burned by our representational choice. There has since developed a number of representational proposals to account for this phenomenon. One (Holyoak & Patterson, 1981) basically assumes that subjects store the absolute position of the objects on a dimension and then make their judgements by *discriminating these* position values with the discrimination easier the further away the items are. Thus, in simple form, the counter proposal might represent the list:

monkey	0.0
tiger	1.0
giraffe	1.5
cow	2.0
donkey	3.0

To judge a pair, the subject would retrieve the positional values and discriminate them. Discrimination was postulated to be a function of these positional values. The representation has the inner items closer together because evidence indicates these are harder to discriminate.

Despite clear case histories of this variety, there has not emerged a clear understanding of what is critical about representential choice. The basic problem is that representational notations such as the ones above are not self-explanatory in that it is not clear what is important about them. Palmer (1978) has argued that a theory of representation needs to make clear what aspects of the representation correspond to what aspects of the represented world. Some aspects of the representation may be just "pure notation" and have no significance. A reasonable question to ask of any representational notation is what aspects of it are critical and what aspects are "just notation". Consider the representation above for the linear ordering. Clearly, certain features like the length of the line between the object and the number are just notation. Our representation would not change if we replaced the lines by hyphens. What about the absolute values of the numbers? Presumably there are some arbitrary aspects here too. Multiplying each by ten, with appropriate changes in the rest of the theory should not result in any predictive changes of the theory. Suppose we chose to represent the words by some number code for each letter (as they are represented in the computer). So we might represent the donkey -- 3.0 association as 4-15-14-11-5-25--3.0. This would certainly obfuscate scientific communication but would not fundamentally change our theory of representation.

However, even when we have pinned down what represents what we really do not have a psychological theory of representation. This is because a representation is a static structure and makes no inherent predictions. Just one example of this is the fact that we can get the correct predictions out of the original Anderson & Bower representation by appropriate choice of process. Wickelgren (1974) basically suggested how this might be done. Suppose that we had an activation process that spread activation from the two anchor points rapidly throughout the propositions (as McKoon & Ratcliff, 1979 have suggested). Further suppose that the activation of the propositions directly attached to the anchor points (1 and 4) was 1.0 and the activation of the inner propositions (2 and 3) was .5. This would reflect a decay of activation from the source. If time to judge the statement was a function of the sum of the activation of the propositions that were needed to be composed to decide, then identical predictions are derived as from the dimensional or positional representation.

There is an emerging consensus (Anderson, 1978; 1983; Newell, 1981; Pylyshyn, 1974; Rumelhart & Norman, 1984; Simon, 1978; Simon & Larkin, in press) that there are two significant issues in a representational theory. One concerns the correspondence aspect of a relationship between the representation and the world it represents. The second concerns the processes that operate on that representation. We will refer to these as the correspondence aspect and the process aspect of a knowledge representation. Two representational systems are different with respect to their correspondence aspect if they do not preserve the same distinctions in the represented world. Thus, if we had a representation that encoded identically the order "A-B-C-D-E" and its mirror image "E-D-C-B-A" this representation would predict that subjects show left-right confusion -- a prediction which neither of the previously mentioned representations is committed to. As another example one could make the positional representation contain more information than the propositional by appropriate assumptions about encoding processes. If the actual numbers could be influenced by factors like length of pauses, then it could encode discriminations in the environment not possible in the propositional representation without some embellishment. So, a very real psychological decision in choosing a representation is what discriminations it preserves among external events.

There are two ways that theories have developed the correspondence between the world and the representation. The more common is a set of assumptions, often informal, prescribing what the correspondence is. However, it is also possible to specify the encoding processes that map stimulus inputs into cognitive representations. In this case the correspondence aspect of a representation becomes a special case of the process aspect of a representation.

While the representation is important, in practice the importance of representational choice turns more on assumptions about the internal processes that operate on the representation. It is really these processes which give a representation its psychological significance. If we choose a set of processes for one representation that are isomorphic to the set of processes for another representation then the two representations really are just notational variants no matter how different the marks on the paper look. On the other hand what might appear to be a trivial superficial difference between representations can be significant if there is a significant difference in the processes that respond to that distinction. So the length of the line in the earlier example (p. xxx) might really matter if it indicated strength of association.

If two representations make the same discriminations among referent states and if they have the isomorphic processes operating upon them, then they represent essentially notational variants of the same theory. However, the notational choice is not insignificant. The major function of notation is communication -- both among scientists and within a scientist (as he or she read their notes to themselves). A representation should be fashioned to make salient the features used by processes defined on that representation and to facilitate analysis of these processes. Choices at this level determine not the predictions of theories, but the ease with which they can be derived and communicated. This is to say that while much of a representational theory is just notation, it is not without substantial consequence.

The evolution from the ACT* representation to the PUPS representation is a case in point. We began working on analogy within the ACT* representation but found that the representation was obfuscating essential features of the analogy process. In reworking the representation we found that we not only clarified analogy but clarified other new issues about knowledge acquisition and clarified some of the older issues in ACT*. So we now are in a position to provide a more cogent analysis of representational issues, new and old.

Analogy in PUPS

PUPS (Anderson & Thompson, in press) is a production system much of the character of ACT* -- it has a declarative memory of factual knowledge which is operated upon by a production system encoding procedural knowledge. For current purposes its key new feature is that it has an analogical mechanism which allows it to derive new declarative knowledge in analogy to old declarative knowledge. Given that the PUPS representation has been principally motivated to facilitate analogy, it is important to explain the basics of PUPS analogy before launching into an extensive discussion of representation. However, an attempt to discuss analogy process first creates a bit of a chicken and an egg problem in that we will be informally using the new representation to describe the analogy process. Then with this analogy process in place, we will turn to a formal development of the complete knowledge representation.

Analogy in PUPS is very much tied to the view of the person as a problem-sover. Analogy tends to be invoked in response to various problem-solving situations. Therefore, the development of analogy will build on examples that are essentially problem-solving in character. The basic idea is that people can solve new problems by analogy to worked-out examples of solutions to old problems. In domains we have studied such a geometry theorem-proving (Anderson, 1982) or LISP programming (Anderson, Farrell, & Sauers, 1984) this seems to be the dominant means that students employ to tackle the solution of novel problems.

Analogical problem-solving in PUPS assumes that problems are represented as forms achieving particular functions under certain preconditions. We can encode this information about examples in schema-like structures where function, form, and precondition are three slots among others in the schema-like representation. Below is illustrated the representation we might want to impose on the example LISP code (+ 2 3)

```
structure1
    isa: function-call
    function: (compute add 2 3)
    form: (list + 2 3)
    context: Common LISP
    medium: CRT-screen
    precondition: context: Common LISP
```

It is represented as a function call that adds 2 and 3 in the context of Common LISP and which was executed on a CRT screen. The form information states that the example is a list structure consisting of the symbols "+", "2", and "3". The precondition information states that it is essential that the context be Common LISP for this form to achieve its function. It would not succeed in INTERLISP, for instance.

There is a basic semantics underlying the relationship among the form, function, and precondition slots. This semantics is that jointly the form and the precondition imply the function. We can represent the example above, for instance, by the following production rule:

```
  IF    goal is to achieve the function (compute add 2 3)
        and the context is Common LISP
THEN    use the form (list + 2 3)
```

that is, implicit in the example is a production rule.

Extending an Example

The basic task of analogical problem-solving is to take an example of a problem solution and extend it to a new situation. This amounts to variabilizing the production rule implicit in the example. For instance, suppose we wanted to add 6 and 3. The production rule we want is

```
IF      the goal is to achieve the function (compute add x y)
        and the context is Common LISP
THEN    use the form (list + x y)
```

In general PUPS wants to extract from examples production rules in which all the terms in the example which fail to match the desired function in the target problem are replaced by variables which do match.

PUPS could always appropriately variabilize the example to satisfy this criteria but at the cost of spuriously extending examples. There has to be a reason to suppose that the example above can be appropriately variabilized without spuriously extending it. Note that in the example above 2 and 3 appear in both the function and the form. This is critical to the induction that replaced them by x and y. The inductive principle underlying this is what we call the no-function-in-identity principle. The basic idea is that it is not an accident that 2 and 3 appear in both function and form. It is their position in the form and not their identity which achieves their function. The analogous function would be achieved by any elements that appeared in the form. Thus. the "no-function-in-indentity" principle allows us to respond to the appearance of a constant value in both form and function of the example by replacing it everywhere by the same variable.

The example above is simple in that it can be transformed into a rule just by variabilizing terms that appear in that structure. However. in most interesting cases of analogy, the terms that appear in the function do not directly appear in the form. Rather it is necessary to elaborate the form and/or function descriptions to come up with a representation that can be variablized. For instance. suppose what we wanted to achieve was to multiply 6 and 3. The above variablized production would have a mismatch between "add" and "multiply". PUPS can get over this hurdle if it has encoded that "+" implements "add" and "*" implements "multiply". encoded by the following PUPS structures:

```
        isa: LISP-function
        function: (implements plus)
        form: (TEXT +)

    *
        isa: LISP-function
        function: (implements multiply)
        form: (TEXT *)
```

One can embellish the representation of the plus example by including this information about the function of the + symbol:

```
  IF    goal is to achieve the function (compute add 2 3)
        and the context is LISP
THEN    use the form (list =function 2 3)
        where =function implements add
```

Note that the "+" has been replaced by the variable "=function" and "=function" has been given the functional description of "+". This reflects the second inductive principle in PUPS analogy, which we call the principle of <u>functional elaboration</u>. The principle is that any term which achieves the function of the replaced symbol will do. This production rule can now be variabilized so that it will extend to the problem of multiplying 6 by 3:

```
  IF    the goal is to achieve the function (compute =op =x =y)
        and the context is LISP
THEN    use the form (list =function =x =y)
        where =function implements =op
```

The constraint "=function implements =op" can be satisfied either by inserting a term which satisfies this function or setting a subgoal to find such a term.

Note that in extracting this production rule from the example and matching it to the problem we have in fact calculated the following mapping of the example onto the problem:

add	--->	multiply
+	--->	*
2	--->	6
3	--->	3

Thus, we have performed the analogy defined by this mapping. However, it is useful to identify the formal rule underlying this analogy. By doing so we more clearly identify the inductive principles. PUPS actually does form and retain these productions as a product of its analogy. These productions are basically proceduralizations in the sense defined in Anderson (1983). That is, they eliminate the need to make reference to a declarative structure in a repeat of a computation. In this case, the data structure eliminated is the PUPS encoding of the example. Subjects do show a dramatic improvement in their problem-solving after using a single example and tend to drop out reference to an example in the subsequent problem-solving episodes (Pirolli, 1985).

Using Analogy to Infer Function from Form

What we have discussed was finding a form to achieve a desired function. One can use the same analogy mechanism to infer the function of a novel form. This amounts to "understanding" the form. The following example, adapted from the dissertation research of Shrager (1985), shows analogy operating in this fashion. Subjects were presented with a tank that had the keypad in Figure 1. They determined that the key labelled with the up-arrow moved the tank forward and they had to figure out what the keys with the down-arrow and left-arrow did. Below we have PUPS structures that purport to represent their states of knowledge:

```
example
    isa: button
    function: (MOVE forward)
    form: (LABELLED up-arrow)

up-arrow
    isa: symbol
    function: (POINTS forward)
    form: (IMAGE thing1)

problem1
    isa: button
    function: ?
    form: (LABELLED down-arrow)
```

```
down-arrow
    isa: symbol
    function: (POINTS backward)
    form: (IMAGE thing2)

problem2
    isa: button
    function: ?
    form: (LABELLED left-arrow)

left-arrow
    isa: symbol
    function: (POINTS leftward)
    form: (IMAGE thing3)
```

The example is encoded as an up-arrow with the further information that an up-arrow is a symbol which conventionally means forward. The functions of the other two buttons are not represented but we have represented the conventional knowledge that down-arrows symbolize backward and left-arrows left.

Insert Figure 1 about here

We can represent the knowledge encoded by the example by the following variabilized production:

```
        IF    there is a structure with form (LABELLED =symbol)
              and =symbol points in =direction
      THEN    the structure has the function (MOVE =direction)
```

This production can be extracted from the example using the "no-function-in-identity" principle and the principle of "functional elaboration" and just switching the form to the condition and the function to the action side of the production. This production enables us to infer that the function of the problem1 button is to move backwards. Similarly we can infer that the function of the problem2 button is to move left. As it turns out, only the first inference was correct. The left-arrow button did not actually move the tank in the left direction but rather only turned it in that direction. This is an example of where the "no-function-in-identity" assumption was violated. Some buttons moved the tank in the specified direction and some turned. One simply had to learn which did which. The actual identity of the direction determined the function of the button. This just proves that PUPS analogy has the danger of any inductive inference. The important observation is that human subjects in Shrager's experiment also made this mis-analogy.

Knowledge Representation in PUPS

Analogy as described is a process that operates on declarative knowledge structures. For its success, it assumes that these knowledge structures are going to predictably contain information about form and function. Thus, use of analogy imposes some well-formedness constraints on the knowledge representation. Moreover, because it creates and stores production rules to summarize the analogy, these well-formedness constraints are propogated to the production rules. Therefore, it should not be surprising that considerations of analogy have played a key role in the evolution from the ACT* knowledge representation to the PUPS knowledge representation.

We will now turn to a formal development of the PUPS knowledge representation. We will essentially achieve this development by presenting a grammar for the syntax of the knowledge representation. As we do so we will describe the semantics and significant psychological correlates of these syntactic distinctions. As forecast in the introduction, these psychological consequences will turn out to be the correspondences between these structures and the external world and the processes associated with the syntactic distinctions. In identifying these consequences we will be in part restating points associated with the ACT* theory and in part stating new PUPS insights. The representation has evolved more drastically than just what would be needed to accommodate the new insights. This extra evolution is part of an effort to clarify the significant issues underlying our representational theory.

In describing the correspondence of these structures to the external world it becomes important first to communicate our conception of the external world. This we refer to as the ontology assumed in the PUPS theory. Basically we view the world as comprised of specific elements such as objects, events, or more abstract entities such as goals. These elements are involved in three types of relationships. First, they can possess a number of one-argument attributes such as color, size, duration, force, etc. Secondly, there are a number of predicates which indicate causal relations between elements. Thus, a goal might cause an action or one event might cause another. Third, these elements participate in constituent relationships such that one element is composed from other elements in some configuration. Thus, a room might be composed by walls, floor, and ceiling in particular configuration. A hockey game might be composed of a sequence of three periods. The knowledge structures we will be defining encode such relations directly or abstractly.

Table 1 presents the rewrite rules of the PUPS knowledge representation. Below we discuss each of these rules separately. The agenda in each discussion is to describe how the PUPS processes respond to the features created in that rule.

Insert Table 1 about here

The Procedural-Declarative Distinction

The procedural declarative distinction is clearly the most fundamental in the PUPS theory and its ACT predecessors with productions defining the procedural component, separate from the long-term declarative component. However, it is another issue what fundamental psychological claims hang on that distinction. It has been known for some time now that a theory which had a long-term memory consisting solely of production rules and just a declarative working memory could do a fair job of mimicking a theory like PUPS which also has a declarative long-term memory. As a simple example, consider the following ACT* example of reasoning about body parts:

Suppose we have stored in declarative long-term memory:

Aristotle was a human.
Humans have color vision.

We assume that while we know these two facts we never have had cause to combine them before. However, we do have a production rule that would combine them on demand. It has the basic form:

```
P1:  IF    the goal is to decide if =obj has =component
           and =obj is in a =category
           and =category have =component
   THEN    =obj has =component
```

Now if "the goal is to decide if Aristotle had color vision" appears in working memory, activation would spread from "Aristotle" and "color vision" to activate the two declarative facts and bring them into working memory and then the production above could be matched and we would add the fact Aristotle has color vision to working memory.

We can get the same effect in a pure production system if we replace the two declarative facts by the following productions:

```
P2:  IF    Aristotle is mentioned in working memory
   THEN    add to working memory the fact that he is human

P3:  IF    color vision is mentioned in working memory
   THEN    add to working memory the fact that humans have color vision.
```

Effectively, we encode in a production rule both the declarative fact and the spreading activation process. Fundamentally, there is no difference between a theory with such retrieval productions and a theory like PUPS. However, the mimicking "all-production" system really has a procedural-declarative distinction built into it. These declarative-retrieval productions are distinct from other productions and basically encode the declarative knowledge. They are distinct because they deposit information into working memory in response to the mere mention of one of the working memory elements. This is what is the distinguishing feature of declarative knowledge -- that it is available in response to mere mention of elements in working memory and does not have its availability restricted to a specific encoded use.

So the basic claim is that whether we have an explicit declarative-procedural distinction in the represen'ation or not there are two types of knowledge which can be distinguished by how the knowledge is treated. This difference in the treatment of knowledge results in a number of phenomena that have been associated with the distinction between declarative and procedural knowledge:

(1) Conscious Report. It is frequently commented that one can report out declarative knowledge and not procedural. This is true whether one has an all-production system or not. In the all-production system the declarative productions deposit a representation of what they know in working memory. This is in contrast to most productions which deposit in working memory a consequence of what they know. Looking at the consequence of P1, "Aristotle had color vision", we are in no position to report the knowledge that gave rise to it.

(2) General Availability. Declarative knowledge is distinguished by the fact that access to it is not restricted to a particular use of the knowledge. This general access is a consequence of the fact that the knowledge appears in working memory response to the mere mention of Aristotle. We do not test for the context in which it appears unlike other productions.

(3) Working Memory Limitation. Because declarative knowledge must be deposited in working memory to be used, limitations on maintaining information in declarative memory

will result in limitations in using declarative knowledge. There is no similar limitation on the procedural knowledge case. This leads to the lesser capacity demands of proceduralized knowledge (Anderson, 1982).

So, in summary, any system which treats the two types of knowledge differently as in PUPS, has a procedural declarative distinction -- whether or not it actually uses these terms and whether or not it uses distinct notational systems for the two types of knowledge. To facilitate scientific development we use distinct names and distinct notation.

The subsequent discussion will first develop the declarative representation and then the procedural representation which is defined in terms of the declarative representation.

Declarative Structures

Declarative memory just consists of a set of declarative structures. As we will also see, declarative structures also turn out to be the principle building blocks in defining the productions that constitute procedural memory A declarative structure has as its *reference* one of the elements that constituted the ontology of the real world that we sketched out earlier. A declarative structure has an internal structure basically to encode the attribute. causal, and constituent relationships we believe it bears in the real world. However. from the outset we want to emphasize our view that declarative structures refer to specific things in the world. One cannot have. in our view. declarative structures which encode general categories of things.

It is a fundamental postulate of our system that declarative knowledge structures describe specific things and not abstractions Thus. on the centuries-old debate between general and specific knowledge representations we come down in favor of the specific The reason for this is two-fold. First. we cannot see any principled way for learning abstract declarative structures. Second. given the abstractions produced by the analogy process and the abstractions encoded into the production systems, this does not appear to be necessary to account for any phenomena that we have been able to think of.

As rule (2) from Table 2 indicates, a PUPS structure consists of an obligatory label, one or more optional types, an optional form, one or more optional function attributes, and preconditions. Form and attribute information were part of the declarative knowledge structures of ACT* where knowledge was chunked into hierarchies of structures where each structure consisted of elements in a particular form. Figure 2 illustrates a few ACT* structures drawn from Anderson (1983) and their rendition in the PUPS representation.

Insert Figure 2 about here

The label is purely a notational convenience for communication on paper. Structures can appear as parts of other structures. Rather than actually embed the structures we put in the structures label. So, for instance, in Figure 2a, the label <u>tall1</u> appears in the form slot of NP1 rather than embedding the <u>tall1</u> structures itself. Thus, labels are effectively pointers in the computer science sense.

Form information is basically information about how the structure is composed and attribute information tells us non-compositional properties. The function and precondition information basically provides an interpretation of the static description contained in these other two elements. The basic claim is that humans have a natural tendency to attribute a function to everything they see. A function is basically a description of the form's position in the causal structure of the universe. PUPS has a set of primitive mechanisms for making causal inferences which record that certain things cause other things. This is recorded in the PUPS knowledge structure by attaching functions to the knowledge structures that encode the form of the things involved in the causation. As we discussed, the PUPS analogy mechanisms reason either from the appearance of forms to what function they might serve or from the existence of needed functions to the shape of forms that will achieve them. As in the case of the procedural-declarative distinction, it is not the terms "form-function" nor the notation that is critical to this knowledge representation. It is how they are used by processes like analogy.

Implicit in almost all theories of human problem solving has been a form-function distinction. These theories have been concerned with how problem-solving operators are deployed to achieve certain goals. Any adequate representation of an operator requires a specification of the form of the operator so that it can be successfully executed and the functions for which it was useful. So, in past research on problem-solving with ACT*, we had already embeded into the theory a form-function distinction. All that we have done in PUPS is to make that distinction explicit in the notation.

The precondition information is tied into the need for discrimination learning in ACT*. Basically, there are certain constraints on forms achieving their functions and the precondition information is a repository for this information. Frequently, this precondition information requires reference to critical attributes of the structure. This was the case in the example earlier where the precondition for the code to work was that the value of the context attribute be Common LISP.

ACT* also required a means for recording discriminating information. In that theory this was done in production memory. So the significant change in PUPS is the change of that discriminating knowledge to a declarative status. This decision is based on data (e.g., Lewis & Anderson, 1985) which indicates that subjects have declarative access to the knowledge which underlies their discrimination.

Types

The distinction between types and functions is purely notational. PUPS treats types as zero-argument functions. The reason we separate it out is we find type information useful in communicating to our audience the basic function of the object while other functional descriptions give more specific information. Thus, we postpone any further discussion until we get to describing the function slots (Rules 9 and 10)

Forms

Every form consists of a template with zero or more arguments (Rule 5 in Table 1). In line with chunking data reviewed in Anderson (1983) and the arguments of Broadbent (1975), we set a bound of five on the number of argument elements that can be combined into a single form. Structures consisting of more elements have to be represented hierarchically with structures embedded as elements within the forms of other structures.

There are two essential properties of this form representation. First, the arguments to the template are ordered. This means that it is not possible to represent directly unordered sets. This assertion is really an assertion about the processes that operate on the contents of the form slots. The assertion is that these processes are position-specific. For instance, in matching a production condition we can match an object in a certain position with a form but we cannot match an object in a position independent manner.

Position-specific matching is a serious restriction in that there are many cases where the knowledge representation would be more powerful if one could represent set structures that were not position specific. As a simple example, addition is a relationship that does not order its arguments and one can reason much more effectively about addition with unordered arguments. *However, the evidence is that children initially order the* arguments of an addition operation and only later learn how to get around the limitations of their cognitive representations and treat addition as an unordered process.

The second critical feature of this representation is that there is a bound on how many elements can be combined in a particular form. Exactly how elements are combined has important consequences because of PUPS' spreading activation retrieval process which was inherited from ACT*. Activation is defined as a property of a structure and spreads from structure to structure through the elements of their forms. Thus, if element A and element B are in the same form they will be activated together and spread more activation from one element to another than if they were separated by being in different substructures. Cognitive psychology is rich in empirical research demonstrating the consequences of such chunking for the availability of knowledge.

One of the things the PUPS theory does not specify, any more than did its ACT predecessors, is how knowledge will be chunked. Knowledge structures are created either by perceptual processes or production firings. The forms created by production firings can be traced back to perceptual encodings. So the point is that we have no theory of how perception will chunk experience although there are a great many people working on the principles of such perception. Certain principles such as the Gestalt principles of similarity and proximity are well-known and serve as rules of thumb for predicting the chunking of specific cases. Thus, we can be fairly confident of how the string XXYYY YYXXX will be chunked independent of any carefully worked out perceptual theory.

Templates

The ACT* theory of representational types lies in the choice of templates (Rule 6 in Table 2). There are at least three representational types in the PUPS. One is the ACT* temporal string which is a simple list of elements. The template identity associated with this type is "sequence". The basic idea here is that one can encode the order of events in an ordinal-type representation which encodes the order of events but not their exact interval positions.

The second type of template is the figure template which encodes a spatial image. As described in the ACT* book, a spatial image is an encoding that preserves the spatial configuration of an array of elements. It is not modality-specific. As such it is not explicitly associated with the visual modality. It is also not size specific. Rather size is an attribute of spatial images. Despite these abstractions, there are potentially an infinite number of spatial images corresponding to ever so subtle distinctions in the placement of up to five elements in a three dimensional array. For the arguments in favor of this structure, consult Anderson (1983?)

We know now no better than in 1983 how to really approach the encoding of spatial images. My solution has been largely one of punting on the issue by creating a different token of the figure template to correspond to each exact spatial image and leaving it up to undefined production system pattern matching apparati to decide when one template is basically a match to another template. Thus, for instance, it is left undefined just what images will be considered as exemplifying a match to the letter A.

The third type of template is the action template by which the system can describe actions such as moving a hand. This is like the motor code that Anderson (1983) speculated might exist. In general such a code is quite critical to describing problem-solving actions. However, we have even less of a theory of such a code than of the spatial code. In PUPS we simply have different action templates to code different actions ignoring such issues as how similarity among different actions might be computed.

The one kind of code that is conspicuous by its absence is a propositional code In Anderson (1983) this was a third representational type. However, as will be discussed at the end of this paper, in PUPS propositional information exists as functional interpretations of the form slots.

The form slot of a structure is supposed to represent a decomposition of experience into elements unaffected by any learned capacity for interpretation. However, propositional representations are learned interpretations par excellence. Most of the predicates we use in propositional encodings (such as buy, read, drive) are derived from experience and are not templates with which our perceptual systems naturally package experience. The form descriptions in PUPS are supposed to represent raw recordings of experience. Their functional descriptions represent learned interpretations.

One interesting consequence of this knowledge representation is that it produces a reduced version of the ontological categories described by Keil (1979). In particular, predicates that refer to the structure of one type of template cannot apply to the others. For instance, predicates about spatial structure (on the right side of X) are only appropriate to those templates; predicates about sequence (the second thing to happen) are only appropriate for those templates, and predicates about agents (is X's fault) are only appropriate for agents.

Note that the arguments in a form slot are pointers (labels) to additional structures, potentially with form slots themselves. The embedding does not go unbounded because at some point we will hit a structure which does not have a form description. This does not mean that it could not have such a description and the potential always exists that PUPS will embellish it with a form description. In some sense the representation assumes that things will decompose into smaller things without limit but this decomposition is always represented in a finite human head in some finite, incomplete form. In PUPS, unlike the original associative analysis of knowledge (for a review see Anderson & Bower, 1973), there are no primitive atoms.

The essential claim in this theory is that it is always possible to represent elements in increasing detail. While this capacity for attaching form descriptions to elements in ever more detail is a particularly transparent way to achieve this, it is by no means the only one. Any representation system which allows for ever more refined descriptions of structures is equivalent to PUPS with respect to this representational assumption.

Attributes

An attribute can be the presence or absence of a value in some dimension (Rules 7 & 8 in Table 1). Again we have by no means a complete listing of the possible attributes of a structure but they include quantity measures like the duration of a temporal interval, the size of a spatial structure, or the force of an action. They can also include features such as color, pitch, location, and time. Basically, in the distinction between form and attribute we have revived Locke's distinction between primary and secondary qualities, something that has been with psychology throughout the generations.

The significance of the distinction lies in the PUPS inference mechanisms which have as a first assumption that the form properties are relevent to the structure's causal properties and only consider attribute information when discriminations are required to limit false generalizations based on just form information. Thus, as a novel first assumption we will believe a tool's shape is more critical to its function than its size, weight, or color although we are prepared to learn otherwise. Any representational theory which imposes an ordering on the properties of an object with respect to causal inference is equivalent to the PUPS theory in this respect.

Function

A functional specification (Rules 11 & 12 in Table 2) consists of a relation and some number of arguments. The arguments, according to the rules above, are just PUPS declarative structures. The interpretation is that the structure whose function this is bears specified relationship to the argument structures.

It is our belief that there is a single primitive relation, cause, and its inverse, caused-by. This is a viewpoint expressed many places over the past 20 years (refs). More complex relationships can be decomposed into a set of causal relationships. For instance, this is how we have represented an example of the LISP function CONS:

```
structurex:
    function (insert a (b c))
    form (LIST CONS ' a ' (b c))
```

There are a number of compositions in this representation but the one of concern here is the two argument insert relation in the function slot. A decomposed relationship would have the structure causing the insertion of a into the list (b c):

```
structurex:
    function (cause eventx)
    form (LIST CONS ' a ' (b c))

eventx
    function (cause (a b c))
    form (putin computer a (b c))
```

where putin in eventx describes an action. Causality is fundamentally a two argument predicate in which one thing causes a second. We already have one thing in the structure whose function we are specifying. Thus, there is only room for specifying one more thing in the function slot. Thus, any system of representation that gives causality the same central role would be equivalent to PUPS here.

The assumption of cause as the sole basic relation means that it is the only one for which we have an innate predisposition to insert between elements of experience. The ability to attribute composed relationships is acquired from use of language as is the ability to recognize composed categories. The basic assumption is that when we have a sentence like "This LISP code inserts a into the list (bc)" we identify the phrases (This LISP code, a, into the list (bc)), the referents of these phrases, and the relational term (insert). We create a function which consists of this relation and arguments which are the referents of the remaining phrases (e.g., insert a (b c))). This *functional description gets attached to the* referent of the sentence subject (e.g., (CONS 'a '(bc))).

The following are the structures that would be created when we add the insert relation as another way of representing the function of structurex:

```
structurex:
          function: (insert a (b c))
          z        : (cause event x)
          form     : (LIST CONS ' A ' (B C))

eventx:
          function: (cause (a b c))
          form     : (putin computer a (b c))
```

By computing form-to-function and function-to-form analogy we would learn the following rule:

```
P1:  IF    structureX
           function: (insert X Y)
  THEN     =structureX
           form: (LIST CONS ' X ' Y)

P2:  IF    structureX
           function: (cause structureZ)
       structureZ
           form: (putin computer X Y)
  THEN     structureX
           form: (LIST CONS ' X ' Y)

P3:  IF    structureX
           form: (LIST CONS ' X ' Y)
  THEN     structureX
           function: (insert  X Y)

P4:  IF    structureX
           form: (LIST CONS 'X ' Y)
  THEN     structureX
           function: (cause structureZ)
       structureZ
           form: (putin computer X Y)
```

Composing 1 and 4 we learn the following rule for structural refinement:

```
P1&P4:     IF    structureX
                   function: (insert X Y)
          THEN   =structureX
                   function: (cause structureZ)
                 structureZ
                   form: (putin computer X Y)
```

Similarly, composing 2 and 3 we learn the following rule:

```
P2&P3:     IF    structureX
                   function: (cause structureZ)
                 structureZ
                   form: (putin computer X Y)
          THEN   =structureX
                   function: (insert X Y)
```

All insert offers is a more compact way of encoding information. Nonetheless, this can be an important feature when we are in situations where successful processing requires minimizing the amount of information to be held active in working memory.

Precondition

The final declarative feature is the potential for various preconditions on a form achieving a particular function (Rule 11 in Table 1). These are basically notes the system takes to itself about the limits in the range of applicability of an example. The form of the note is simply for the system to record that a particular piece of declarative structure is critical for this form achieving its function. Thus sitting in the precondition slot is just a piece of declarative structure.

The following is a classic example of how a system should represent to itself how to arrange for block2 to be on block1:

```
action
   isa: move
   function: (achieve stack block1 block2)
   form: (move person block1 topblock2)
   precondition: (topblock2 attribute clear)

block1
   isa: block
```

```
        form: (block topblock1 bottomblock1...)

    block2
        isa: block
        form: (block topblock2 bottomblock2...)

    topblock2
        isa: surface
        attribute: clear
```

where topblockx and bottomblockx refer to various sides of blockx.

Essentially, we have represented a person moving block1 to top of block2 to achieve the state of block1 on block2. However, there is a precondition represented -- that the part of block2 which is its top must have the attribute of being clear. Storing this precondition will prevent this action from inappropriately being used as an anological model in future problem-solving.

Perhaps the most critical feature of this way of representing discriminating information is that it is declarative and something available for reporting. Thus, this representation has embedded the claim that we can *identify the features that are critical* to a *form achieving* its function. One of the major criticisms we had of the ACT* theory was that it did have conscious access to the discriminating features it was acquiring whereas subjects appeared to have such access (Anderson & Lewis, 1985). Any knowledge representation that embeds this claim is essentially isomorphic to PUPS on this score.

Negation

Note that attributes, functions, and preconditions can be optionally negated (Rules 7, 9, and 11 in Table 1). A negated attribute encodes that a structure does not have a particular attribute. Thus, we might note of a dog that it doesn't bark. A negated function encodes the fact that a form negates a certain function. Contraceptives would be represented with certain obvious negating functions. A negated precondition encodes the fact that a certain declarative structure must be absent for a form to achieve its functions.

Procedural Structures

Procedural knowledge in PUPS (Rule 13) takes form of the traditional production rule of a condition and an action. Basically, if the pattern described in the condition exists in working memory the data structures specified in the action will be added to working memory.

The production rule embodies a certain piece of knowledge just as does the declarative fact. The essential difference between the two is the asymmetry imposed by the condition-action structure. Knowledge evoked in one circumstance where the condition matches need not be available in another. For instance, McKendree and Anderson found subjects who learned how to evaluate LISP expressions but could not turn that knowledge around for purposes of generation of LISP expression to achieve desired evaluations. Declarative knowledge, on the other hand, can be equally evoked in all circumstances. The advantage gained by the inflexibility of procedural knowledge is efficiency.

Condition

The condition of a production is just a description of a set of declarative structures (Rule 13 in Table 1). Basically, it is significant that production conditions make no reference to anything but standard declarative structures. In particular, there cannot be reference to things like goal structures as there was in ACT*. This in PUPS is the embodiment of the claim that the control of behavior lies in the structures that the behavior is operating on.

It is also significant that the condition is basically a conjunction of tests for the presence or absence of declarative structure. There cannot be any more complex Boolean functions. As one instance, suppose we wanted to recognize that A was B's uncle. The following is the predicate we might want to write:

```
IF  OR [  {= A: function (brother-of = C)}
          {= A: function (husband-of = D)
           = D: function (sister-of = C)}]
    OR [  {= C: function (father-of = B)}
          {= C: function (mother-of = B)}
THEN =A: function (uncle-of =B)
```

This production has compressed into it the four possible ways A can be B's uncle. It is a conjunction of two OR's which is not allowed in PUPS. Rather PUPS would have to have four separate productions to recognize the four possibilities. The significance of this representational claim is that the child would have to learn separately the four possible rules for classifying uncle.[1] Thus, this feature of PUPS productions produces the well documented conjunction bias in human cognition (ref.)

Another important feature of these production rules is the way they treat negation. The following would be a legal production condition (or component of a condition):

```
=person
   -health:sick
```

This tests for the existence of a person of whom it is explicitly stored that the person is not sick, (i.e., the person is healthy). Thus, we can explicitly test for explicitly negated structures in memory. This is the only kind of explicit negation test that we can have. What we cannot do is explicitly test whether it is not known that the person is sick. That is, we cannot explicitly test for implicit negation in the data base (i.e., knowledge negated because it is absent). The only way to test for implicit negation is by conflict resolution among two productions. So, if we had the following three productions,

```
production1
   =person1
         health:sick
         status:queried
   ---->
```

[1]This, of course, is only an accurate claim under certain representations. If we had a predicate like <u>parent-of</u> we would avoid the need for separate <u>mother</u> and <u>father</u> productions.

```
production2
    =person2
            -health:sick
            status:queried
    ---->

production3
    =person3
            status:queried
    ---->
            say "don't know"
```

Then productions 1 and 2 would take precedence over 3 because they are more specific. Thus, production 3 would only fire if 1 or 2 did not match. Thus, in PUPS we can only implicitly test for implicit negation. Consult Anderson (1983, Chapter 3) for evidence for this way of making absence judgements.

Variables

The one way that the data structures in productions differ from the data structures in declarative memory are variables (rule 14 in Table 1), the symbols prefixed by "=". These are elements introduced by the no-function-in-identity principle of analogy or the principle of function elaboration. The *no-function-in-identity* principle causes a variable to replace a constant when the same constant would appear on the left and right hand side of a production. The principle of functional elaboration creates variables when a constant has its functional structure specified. In either case variables essentially permanently encode in the production rule the generality assumed by the analogy process.

There are two ways variables can be treated in pattern matching. We can either allow the same constant to match to more than one variable or not. As in the ACT* pattern-matcher we allow the same constant to bind to more than one variable. This means if we learn a rule about subtracting two members of the set:

```
   IF   the goal is to subtract =x from =y
        and =x + =z equals =y in the addition table
 THEN   the answer is =z
```

This rule will apply even when =x and =z happen to bind to the same number. It seems essential that we treat variables in this way for production like the above rule to work.

Action

As is the case with the condition of a production, the action is just a set of regular declarative structures with no special embellishments (Rule 15 in Table 1). Perhaps the way to appreciate the significance of this is to consider the things that might have been part of a production action (and indeed are in other production languages) but which are not in PUPS:

Deletion. Many production languages allow one to delete structures from working memory. This can be a very effective device for the purposes of avoiding clutter in working memory and controlling flow of the production system. PUPS continues the ACT* tradition of an add-only memory. Read references for the arguments that this is a feature of human memory.

Goal Manipulation. While ACT* had no facility for deleting regular structure it could reset pointers in a goal stack (resetting amounts to a deletion followed by an add). This is not part of PUPS, since it has no goal stack. Basically, ACT* was too inflexible in its order of processing information. As Van Lehn (ref) has documented for the domain of subtraction and as seems true in other domains people can show considerable flexibility in the order in which they execute a well-learned skill.

Direct Actions. Many production systems can take direct actions in the external world as well as add to working memory. This is not possible in the PUPS architecture. Production system actions can deposit requests for action in working memory which can be acted upon by motor routines. However, the buffering of these action requests in working memory gives the system a moment to reflect on what it is about to do and to abort these intentions if that seems wise.

As in the condition side of productions, the one difference between the syntax of production actions and regular declarative structures is the possibility of variables. Variables in the actions that occur on the condition are replaced by the constants they bind to in the condition. When a variable occurs in the action and not in the condition a new token structure is created and introduced as the value of the variable. So, to be concrete consider the following production:

```
IF      =person1
        function: (grandfather-of =person3)
THEN    =person1
        function: (father-of =person2)
        =person2
        function: (parent-of =person3)
```

This is a production that makes the inference that if =person1 is the grandfather of =person3, then there must be an intervening =person2 who has person1 as a father and =person3 as a child. The point is that =person2 is a variable which only appears in the action of the production. In such cases a new structure node is created and assigned as the value of the variable.

One of the dangers in such creations is that the reference of the variable is already represented in memory and we have created an additional copy. Thus, we may know =person3's mother and this may happen to be the =person2 in question. This is an almost unavoidable circumstance in dealing with such a memory -- that multiple tokens will be created with the same referents. In research on the topic (Anderson, ref.) it was found that people have great difficulty in such situations and that they simply cannot merge nodes with similar referents. Rather they have to be prepared to reason about the identity of referent when they discover it. Dealing with such circumstances is one of the principle functions of the identity predicate described earlier.

Propositional Network Representations?

The preceding concludes our definitional discussion of the PUPS representation. There remains one point that is sorely in need of discussion: Where are the propositional networks that have been such an integral part of the ACT* representational discussions? Indicative of a fundamental switch in the PUPS theory to a more "peripheralist" interpretation, there is not a propositional type for form slots to complement the temporal string, spatial image, and action category. The form slots are supposed to be encoding of direct experience and we do not directly experience propositions. Rather propositions are interpretations of experience.

First let us say a few words to dispose of the minor issue of networks before turning to the critical issue of propositions. The notation we have been using in this paper has been linear purely for convenience. Were we to get into a discussion of issues such as spreading activation we would turn to a network notation. As argued in Anderson (1985 and elsewhere) the choice between network and linear representations is purely a matter of convenience for the current analysis.

With respect to the more substantial issue. the PUPS representation does not conflict with the mass of evidence that has been marshalled for the existence of a propositional representation. This is because propositional information exists in the function slots that serve as the interpretation of the forms. Thus the issue is how we represent what was called propositional information and not whether we represent such information. The following is how we would represent that classic sentence "The boy hit the girl":

```
Sentence
    function: (communicate event)
    form: (LIST subject hit object)

subject
    function: (communicate agent)
    from: (LIST the boy)
```

```
object
    function: (communicate patient)
    form: (LIST the girl)

event
    function: (hit agent patient)
    form: (ACTION-hit agent patient hand)
```

Thus, the sentence is interpreted as describing an event involving the action of agent hitting patient with his hand. Thus, we relate this surface sentence with a surface action. However, with the event we also have the <u>hit</u> function derived from the use of hit as a relational term in the language. Thus, we also have a connection between the sentence and propositional information in the function slot.

One class of evidence for propositional representations is that under many circumstances our interpretations of events better than the actual surface events (refs). Under the current analysis this comes down to the claim that we are more likely to remember the function slot of an event than its form slot.

Consider the implications of this position for Paivio's dual code theory (1969) which the propositional theory developed in contrast to. According to dual code theory, one represents information either verbally or visually. Our current analysis gives us something in the same peripheralist spirit but the details are different. The differing details include the added action data type but more importantly the fact that the temporal-string representation is not tied to the verbal media and the spatial image is not tied to the visual media. In fact much of the evidence against the dual code theory turned on tieing these representations to a particular sensory or linguistic medium. In more recent publications Paivio (ref) has accepted an interpretation of a dual code theory that does not strongly tie it to a modality (ref.).

Another major category of evidence against the dual code theory is the fact that we can remember the meaning of abstract assertions which do not have any obvious external reference. To take one example from Anderson and Bower (1973), we can represent the distinction between someone forging a check and someone signing a check although these two events might be exactly identical in terms of the actions observed. It is worthwhile to examine this example in detail within PUPS, both to show how it interprets the abstract subtleties in a predicate like "forge" and to show the critical role analogy has in the ACT theory of meaning.

Table 2 shows a PUPS encoding of the sentence "The banker forged his signature on the paper". Figure 3 provides a partial network representation of this information.

Insert Table 2 and Figure 3 here

The sentence is encoded as communicating an event which consists of two steps. The function of the event is to get a signature (called scribble) on a location on a paper. The first step, step1, is further decomposed *into two substeps, step11 and step12. Step11* involves the action of the banker grasping the pen and step12 involves moving the pen to the paper. Jointly they achieve the intermediate state of the pen being at the paper. Step2 involved the banker moving the hand at a position on the paper (paperspot) in order to create the signature. As described this contains no more information than would be in an uninterpreted visual image of a banker writing a signature. In particular it does not distinguish between a true signature and a forgery. However, when we look at the description of the signature (scribble) we see the information that separates a forgery from a true signature. Scribble is not equal to the signature that is possessed by the banker Rather it is encoded as being the signature that is possessed by some other person not identical to the banker. All this information is contained in the functional elaborations of the two signatures and the two people involved. In general any such abstract information can be contained in the functional elaboration of terms. This reinforces the point made earlier that propositional information is not a data type of the same character as an image. Rather it exists as a functional interpretation of an image.

One might ask of such a knowledge representation exactly how it encodes the meaning of "forge"? There is no abstract data structure sitting around unpacking the meaning of forge. Rather there is the concrete word "forge" with pointers to its occurrence in specific sentences and representations of the situations described by these sentences. There is also the derived relation forge in the functional description of event, but it is only a summary for these specific situations. Still this serves as an adequate basis for understanding new sentences because new sentences can be interpeted by analogy to the representations created in the interpretation of old sentences.

For example, suppose that the sentence "The lawyer forged his signature on the check" is encountered. When PUPS was presented with this sentence it proceeded through the following steps of analogical interpretation:

1. The function of this sentence was also to describe an event which
   a. achieves a state
   b. such that there is a signature on the paper
   c. that is equal to the signature of person1 but not person2
   d. where person2 is the lawyer
2. The structure of the event is that there are two subevents where
   a. the first event achieves the state of having the pen at the paper
   b. the second event achieves the goal of the main event.
3. The second event involves the lawyer putting the signature on the check
4. The first event involves a sequence of two subevents where
   a. the first subevent achieves the goal of the lawyer possessing the pen

b. the second subevent achieve the goal of the first event

5. The second subevent involves the lawyer moving the pen to the paper.

6. The first subevent involves the lawyer grasping the paper.

It is not necessary to go through this elaborate analogy process every time the word "forge" is encountered. Instead we could attach to the described event a functional description with the relation forge. This would provide a more compact way of encoding experience which could be expanded upon demand into more decomposed forms as in Figure 3. Thus, our theory of representation has a sort of Worfian character to it. That is to say, the terms with which we represent experience are often derived from the words in natural language. However, as this example shows it is not the case that this in any way changes the distinctions we are capable of encoding. Rather, it only provides a convenient shorthand for communication otherwise complex information.

Conclusions

In concluding this discussion of PUPS representation it is worth remarking on how radically different it appears from the earlier ACT representations. However, these differences are largely superficial. The representational notation has been refashioned to facilitate discussion of analogy and causal inference, which are two prime learning mechanisms in PUPS which were not in ACT*. However, this refashioning has been for ease of exposition and development. There is no part of these learning mechanisms that could not have been supported in ACT*. Indeed we basically worked out the simulation of analogy in GRAPES, which is a production system that simulates ACT*, before developing the PUPS representation to help clean up the implementation of analogy.

It is to the credit of the representation that it facilitates the development and exposition of the PUPS theory. However, this should not obscure the fact that the development from ACT to PUPS has not changed most of the fundamental claims we are making about how knowledge is used in cognition.

Table 1
Grammar of PUPS
Knowledge Structures

(1) Knowledge	----> Procedural
	----> Declarative
(2) Declarative	----> Label + (Type*) + (Form) + (Function*) + (Attribute*) + (Precondition*)
(3) Type	----> isa Category
(4) Category	----> sequence, event, object, action, scene, symbol, agent, desire, etc. plus composed categories
(5) Form	----> Template + (label*)
(6) Template	----> Sequence
	----> Figure
	----> Action
(7) Attribute	----> (-) Dimension + value
(8) Dimension	----> length, size, force, color, pitch, time, location, etc.
(9) Function	----> (-) Relation + (Label+)
(10) Relation	----> cause, composed forms
(11) Precondition	----> (-) Declarative
(12) Procedural	----> IF Condition THEN Action
(13) Condition	----> Declarative*
(14) Label	----> Variable/Procedural
(15) Action	----> Declarative*

Notes

- parentheses indicate optimal elements
- * indicates the possibility of more than one instance
- / indicates the rewrite rule only applies in the following context.

Table 2

```
sent
        isa: sentence
        function: (describes event)
        form: (list subject word object location)
subject
        isa: nounphrase
        function: (communicate person1)
        form: (list word1 word2)
word1
        isa: word
        form: (text the)
word2
        isa: word
        function: (bankerhood)
        form: (text banker)
word
        isa: word
        form: (text forged)
object
        isa: nounphrase
        function: (communicate scribble)
        form: (list word3 word4)
word3
        isa: word
        form: (text his)
word4
        isa: word
        function: (signaturehood)
        form: (text signature)
location
        isa: prepphrase
        function: (communicate paper)
        form: (list word5 word6 word7)
  word5
        isa: word
        function: (onhood)
        form: (text on)
  word6
        isa: word
        form: (text the)
  word7
        isa: word
        function: (paperhood)
        form: (text paper)
  event
        isa: mainevent
        function: (achieve state)
                    (forge person scribble paper)
        form: (sequence step1 step2)
```

```
state
     isa: state
     function: (result-of event)
     form: (at place scribble)
step1
     isa: step-sequence
     function: (enable step2)
               (achieve state1)
     form: (sequence step11 step12)
step11
     isa: step
     function: (enable step12)
               (achieve state11)
     form: (grasp person1 hand pen)
state11
     isa: state
     function: (result-of step11)
     form:: (in pen hand)
pen
     isa: pen
     function: (enable writing)
     form: (text pen)
hand
     isa: hand
     function: (possessed-by person1)
     form: (text hand)
step12
     isa: step
     function: (achieve step1)
     form: (move-to person1 hand paper)
state1
     isa: state
     function: (result-of step1)
     form: (at pen paper)
paper
     isa: paper
     form: (text paper)
step2
     isa: step
     function: (achieve event)
     form: (move person1 hand paperspot)
paperspot
     isa: position
     form: (text paperspot)
scribble
     isa: writing
     function: (inequality signature1)
               (equality signature2)
     form: (text scribble)
signature1
     isa: signature
     function: (possessed-by person1)
```

```
      form: (text signature1)
signature2
      isa: signature
      function: (possessed-by person2)
      form: (text signature2)
```

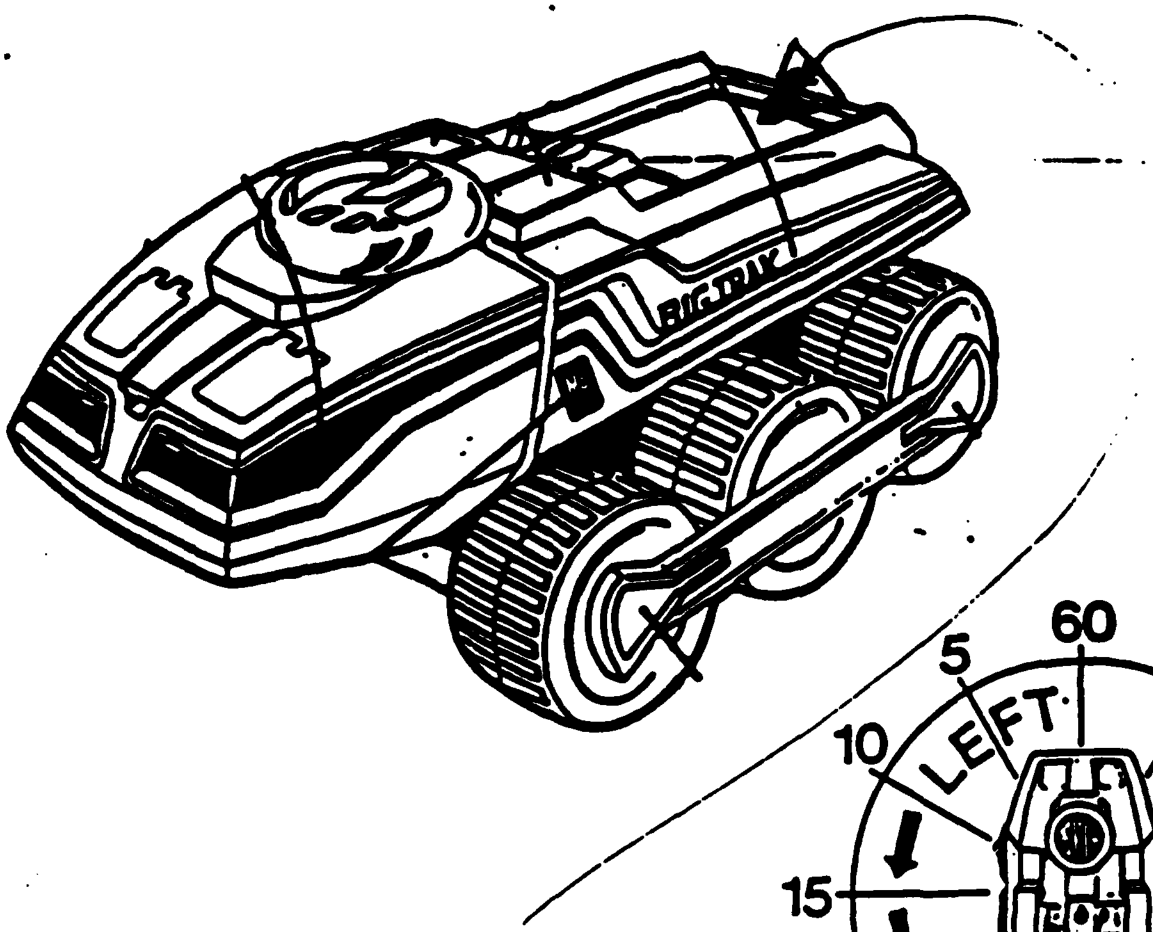
Figure Captions

Figure 1. Experimental situation used by Shrager (1985).

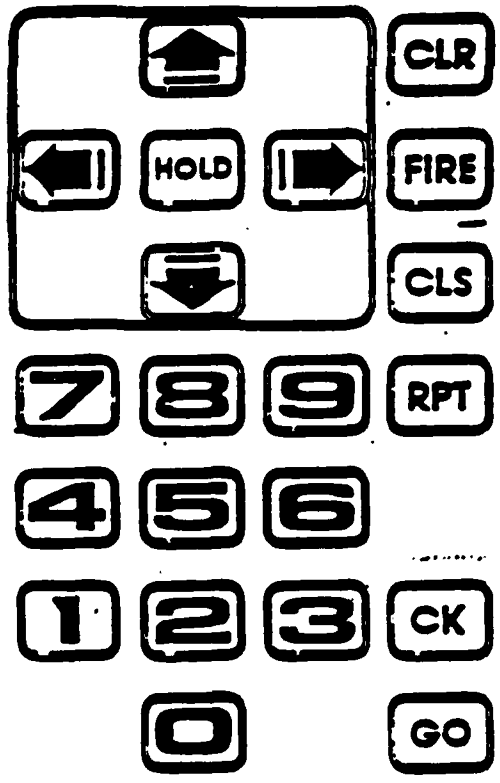
Figure 2. Examples of knowledge representations in ACT* and their corresponding representations in PUPS.

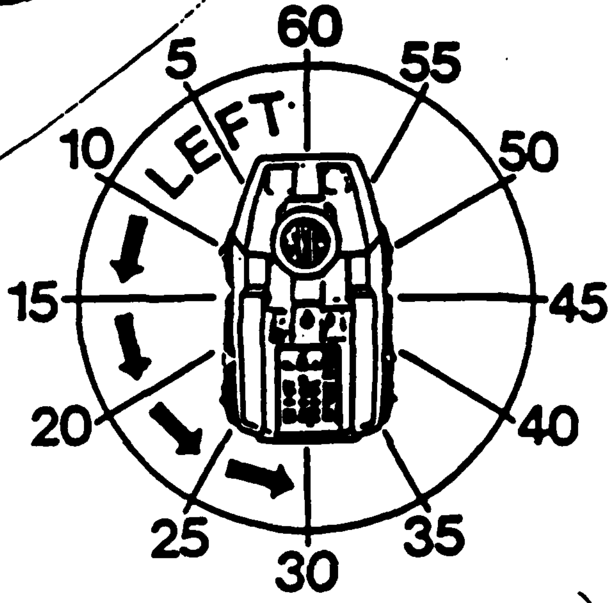
Figure 3. A partial representation of the information in Table 2.

a)


BIG TRAK

b)


HOLD
CLR
FIRE
CLS
RPT
7 8 9
4 5 6
1 2 3
CK
0
GO


LEFT
60
5
10
15
20
25
30
35
40
45
50
55

c)

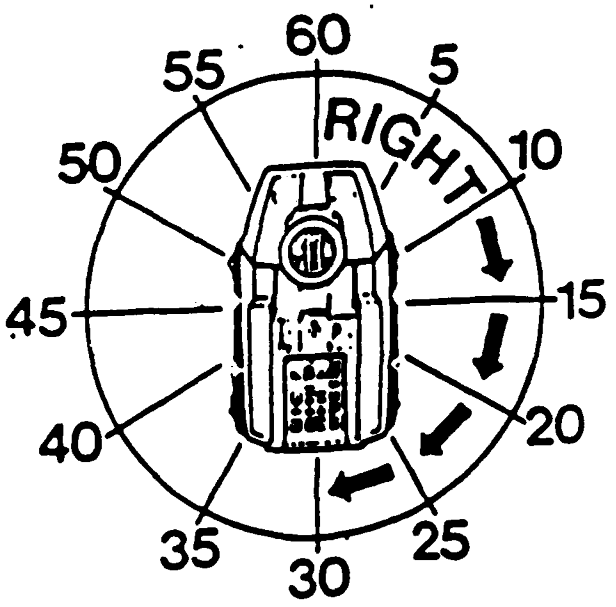

RIGHT
60
5
10
15
20
25
30
35
40
45
50
55

Figure 1

Figure 2a

NP1

isa: nounphrase
function: (describe Fred)
form: (sequence the1 tall1 young1 man1)

tall1
isa: adjective
form: (sequence T AH)

young1
isa: adjective
stress: positive

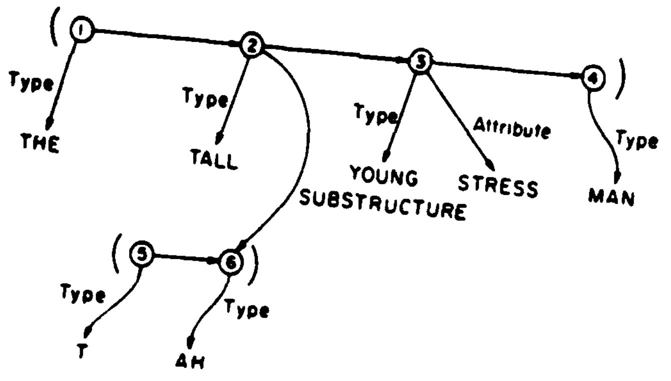


Figure 2b

A1

isa: array

function: (test subjects santa-experiment)

form: (face-array triangle1 square1 circle1)

size: 100

triangle1

isa: triangle

size: 2

square1

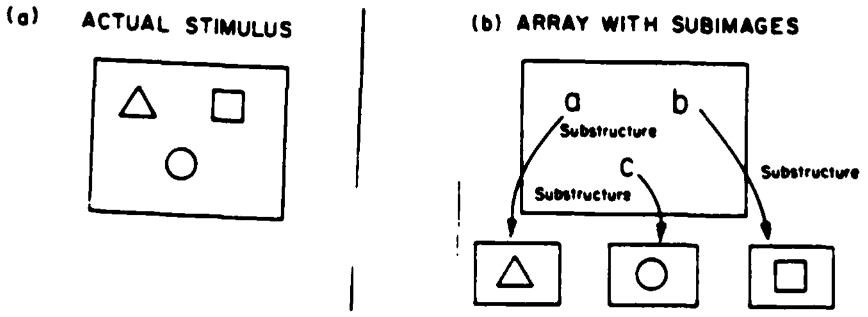
isa: square

size: 4

circle1

isa: circle

size: 3



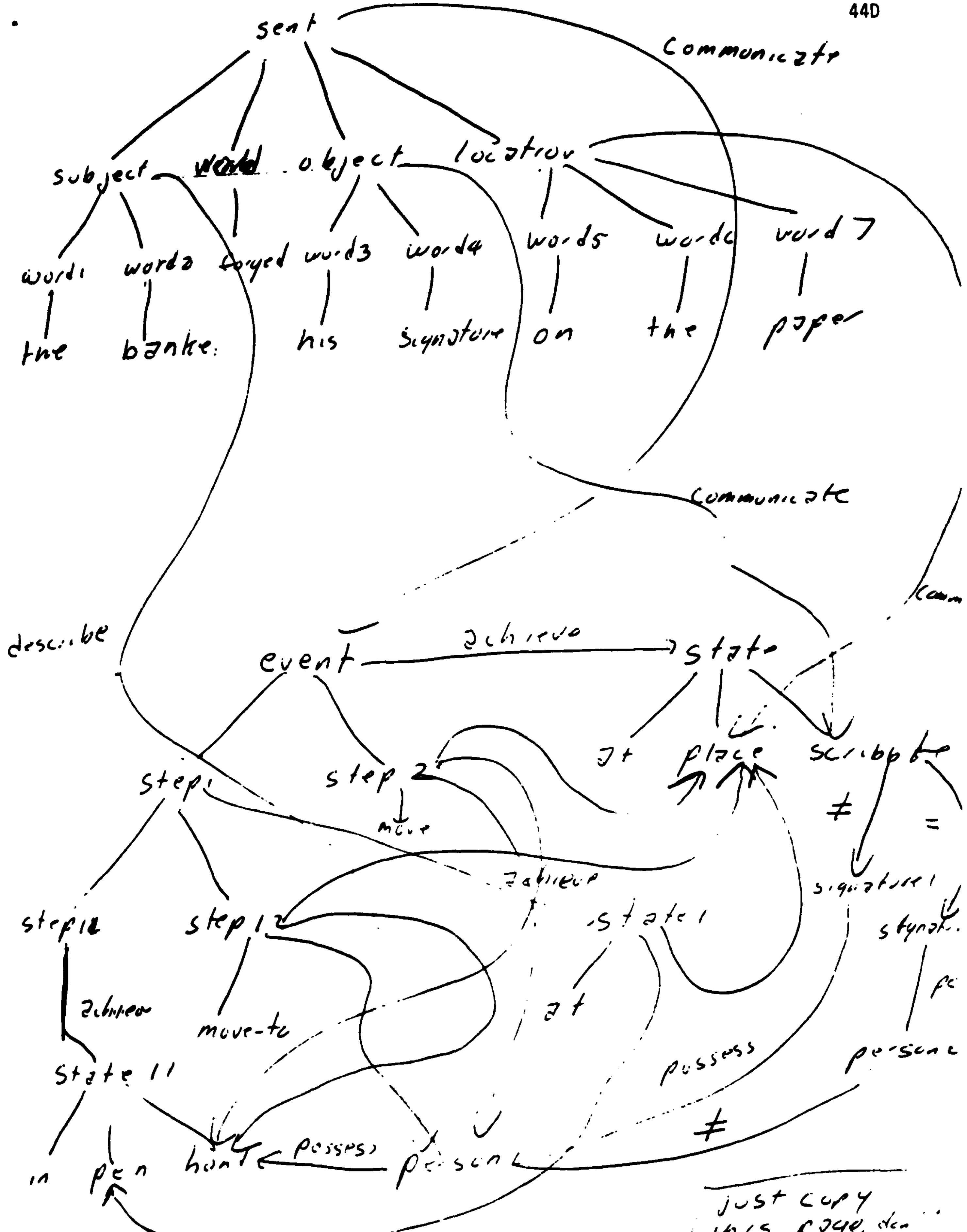


Fig 3

just copy this page, don't attempt to redraw now

References

Anderson, J.R. (1976). *Language, Memory, and Thought.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Anderson, J.R. (1978). Arguments concerning representations for mental imagery. *Psychological Review, 85,* 249-277.

Anderson, J.R. (1982). Acquisition of proof skills in geometry. In J.G. Carbonell, R. Michalski & T. Mitchell (Eds.), *Machine Learning. An Artificial Intelligence Approach.* Palo Alto, CA: Tioga.

Anderson, J.R. (1982). Acquisition of Cognitive Skill. *Psychological Review, 89,* 369-406.

Anderson, J.R. (1983). *The Architecture of Cognition.* Cambridge, MA: Harvard University Press.

Anderson, J.R. (1985). *Cognitive Psychology and Its Implications. Second Edition.* New York: Freeman.

Anderson, J.R. & Bower G.H. (1973). *Human Associative Memory.* Washington, DC: Winston and Sons.

Anderson, J.R. & Thompson, R. (in press). Use of analogy in a production system architecture. In A. Ortony, et al. (Eds.), *Similarity and analogy.* ?

Anderson, J.R., Farrell, R., & Sauers, R. (1984). Learning to program in LISP. *Cognitive Science, 8,* 87-129.

Broadbent, D.E. (1975). The magic number after fifteen years. In R.A. Kennedy & A. Wilkes (Eds.), *Studies in Long-term Memory.* New York: Wiley.

Kosslyn, S. M., & Pomerantz, J. R. (1977). Imagery, propositions, and the form of internal representations. *Cognitive Psychology, 9,* 52-76.

Lewis, M.W. & Anderson, J.R. (1985). Discrimination of operator schemata in problem solving: Learning from examples. *Cognitive Psychology, 17,* 26-65.

McKoon, G. & Ratcliff, R. (1979). Priming in episodic and semantic memory. *Journal of*

Verbal Learning and Verbal Behavior, *18*, 463-480.

Newell, A. (1981). The Knowledge Level. *AI Magazine*, *2*, 1-20.

Paivio, A. (1969). Mental imagery in associative learning and memory. *Psychological Review*, *76*, 241-263.

Palmer, S.E. (1978). Fundamental aspects of cognitive representation. In E. Rosch & B. Lloyd (Eds.), *Cognition and categorization.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Pirolli, P.L. (1985). *Problem solving by analogy and skill acquisition in the domain of programming.* Doctoral dissertation, Carnegie Mellon. Ph.D. Dissertation.

Potts, G. R. (1972). Information processing strategies used in the encoding of linear orderings. *Journal of Verbal Learning and Verbal Behavior*, *11*, 727-740.

Pylyshyn, Z. W. (1973). What the mind's eye tells the mind's brain: A critique of mental imagery. *Psychological Bulletin*, *80*. 1-24

Shrager, J.C. (1985). *Instructionless learring: Discovery of the mental device of a complex model.* Doctoral dissertation. Carnegie Mellon. Department of Psychology.

Simon, H.A. (1978). On forms of mental representation. In C. Wade Savage (Eds.). *Perception and cognition: Issues in the foundation of psychology.* Minneapolis. MN: University of Minnesota Press.